# USING DYNAMIC WFST COMPOSITION FOR RECOGNIZING BROADCAST NEWS

*Diamantino Caseiro, Isabel Trancoso*

$L^2F$ Spoken Language Systems Lab.
INESC-ID/IST
Rua Alves Redol 9, 1000-029 Lisbon, Portugal
{dcaseiro, Isabel.Trancoso}@l2f.inesc-id.pt

## ABSTRACT

Our first application of weighted finite state transducers to the recognition of broadcast news provided us with an interesting framework to study several problems related to the optimization of the search space. The paper starts by describing how the use of our lexicon and language model "on-the-fly" composition algorithm is crucial in extending the transducer approach to large systems. We present an efficient representation for *WFSTs*, that allowed us to reduce runtime memory requirements, and discuss several types of language model optimizations, including a context-sharing algorithm. Experimental results obtained with the broadcast news corpus collected for European Portuguese illustrate the impact of the various possible optimizations of the components on the performance of the system.

## 1. INTRODUCTION

The integration of knowledge sources in large vocabulary continuous speech recognition using weighted finite state transducers (*WFSTs*) is spreading in the speech recognition community.

Among the main advantages of the approach relative to traditional systems are: the elegant and uniform formalism that allows very flexible ways of integrating multiple knowledge sources, and the superior search performance obtained when the search network is optimized using automata determinization and minimization.

The main disadvantages are related to the search space optimization, where we may identify 3 main problems:

- The *WFST* determinization algorithm, based on subset construction, and normally used during optimization, requires large amounts of memory relative to the size of the resulting *WFST*.
- Although the optimized search network is not much larger than the language model (typically only 2 to 2.5 times larger), it can still be very large and requires large amounts of memory in runtime.
- The fact that the optimization of the search network is performed offline means that the original knowledge sources are not available at runtime. Thus, it may be troublesome to preserve the optimality of the network, when dynamically adjusting the knowledge sources. For example, when adapting the language model probabilities or when adding new words or pronunciations to the vocabulary.

The most common solution proposed for the first two problems consist of reducing the size of the system to the resources available during development or run time, and rely on an additional pass to re-score with a larger language model. This approach is effective, but limits the application of the *WFST* approach, to small first-pass systems. Along the same lines, but relying on a single pass, is the approach of Dolfing and Hetherington [1], where the full language model ($G_f$) is factorized into a small model ($G_s$) and a difference model ($G_{f-s}$) such that $G_f = G_s \circ G_{f-s}$; the small language model is incorporated with the other knowledge sources, the lexicon $L$ and the acoustic model $H$, in a network ($N = H \circ L \circ G_s$) that is optimized offline. The full language model information is integrated in runtime by composing the network with the difference language model ($N \circ G_{f-s}$) "on-the-fly".

Solutions to the third problem typically use "on-the-fly" composition to combine the optimized static search network with the dynamic knowledge sources.

In the following sections we will show how we address these problems in our system. We shall start by describing how the use of our lexicon and language model "on-the-fly" composition algorithm is crucial in extending the *WFST* approach to larger systems. We will proceed by showing a memory-efficient representation for *WFSTs*. In section 4, we discuss the language model optimizations and present a context-sharing algorithm for language models represented as *WFSTs*. In section 5 we will describe and discuss recognition experiments performed using our system. The final section summarizes the main conclusions.

## 2. LEXICON AND LANGUAGE MODEL COMPOSITION

The determinization of the composition of the lexicon $L$ with the language model $G$ is probably the most resource intensive subtask when optimizing the search network. The reason lies on the large size of the language model, and on the fact that, when applying the subset-construction determinization algorithm, every state of the resulting *WFST* ($det(L \circ G)$) corresponds to a *set* of states of the non-deterministic $L \circ G$ transducer.

In [2] we presented a memory-efficient specialized algorithm for the composition of the lexicon with the language model. Our algorithm is based on Mohri's theorem [3] that states that the composition of sequential transducers is also sequential. This important result means that if we determinize both the lexicon and the language model, then we only need to compose them to obtain the deterministic composition. In practice, we cannot just apply the

The present work is part of Diamantino Caseiro's PhD thesis, initially sponsored by a FCT scholarship (PRAXIS XXI/BD/15836/98). This work was also partially funded by IST-HLT European program project ALERT. INESC-ID Lisboa had support from the POSI program of the "Quadro Comunitario de Apoio III".

usual composition algorithm [4], because of $\epsilon$ labels on the output tape of the lexicon, which will generate so many non-coaccessible[1] paths in the result, as to make the method unpractical.

Our algorithm is just a specialized composition algorithm, and works as follows: in a preprocessing step, the set of reachable non-$\epsilon$ output labels is associated with each $\epsilon$-output edge of the lexicon. That set is used during composition to avoid the generation of non-coaccessible paths by only following $\epsilon$-output edges on the lexicon that will lead to a non-$\epsilon$ label compatible with labels in the language model state.

This basic algorithm was also extended to allow output-label and weight pushing. In [5] we showed how to extend the algorithm to approximate "on-the-fly" minimization.

When used with a caching scheme, the overhead of performing both the $LG = L \circ G$ specialized composition and the $H \circ LG$ composition in runtime, is only 20% of the search effort[2].

The integration of dynamic information in runtime is easy as both the original lexicon and language model are available in runtime. Our composition algorithm has the added advantage of optimizing also the new information.

## 3. EFFICIENT REPRESENTATION OF *WFSTs* FOR ASR

Even when fully optimized, the *WFSTs* used in large vocabulary tasks can be very large. In order to reduce the runtime memory requirements, when using a large static network, we developed a memory efficient representation for *WFSTs*. Our basic representation is based on an adjacency-list representation for the *WFST* graph. The main data structure of that representation is a vector, containing all the edges, sorted by origin state. Each edge in the vector is a 5-tuple containing: the identification of the origin state; the destination state; the input label; the output label; and the weight. This tuple occupies 20 bytes of memory. Another vector contains the 4-byte offset to the first edge that leaves a particular state. Thus the memory required by the basic graph representation is $4|Q| + 20|E|$ bytes, where $|Q|$ is the number of states and $|E|$ the number of edges.

The new compact representation uses a variable length representation for edges that takes into account the properties of the typical integrated network. The main problem with a variable length representation is that is it more difficult to directly access the edges that leave a given state, as direct indexation would still require 4 bytes per state (for a pointer or an offset). We reduced the memory required per state to 16.5 bits, by grouping the states in groups of 64 states. A master index $M$ contains a 32-bit pointer to a chunk of memory containing the edges of 64 states. A separate offset index $O$ contains the 16-bit offset of the first edge of the states in the chunk. Thus, the first edge of state $q$ is the $O[q]$ edge stored in the chunk $M[q/64]$. The indexation scheme is illustrated in figure 1.

Each edge contained in a chunk occupies an integral number of bytes that can vary from 1 to 10. A variable-length encoded edge is a sequence $FDLW$, where:

- $F$ is a record of 5 bits that specifies how the edge is encoded: 2 bits specify the encoding of the destination state; 2 bits the encoding of the input and output labels; 1 bit specifies the encoding of the weight.

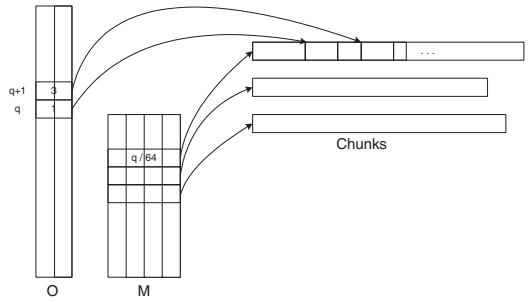


**Fig. 1**. Compact *WFST* Indexation Scheme.

- $D$ encodes the destination. The destination is encoded as the difference between the origin and destination states of the arc. It can be encoded in 3, 11, 19 or 27 bits. 3 of the bits are stored in the same byte as $F$.
- $L$ encodes the input and output labels. If both the input and output labels are $\epsilon$, it is encoded in 0 bytes; if the output is $\epsilon$ and the input is less than 256 it is encoded in one byte; if the output is less than 65536 and the input is $\epsilon$, it is encoded in 2 bytes; otherwise the input and the output are stored in 4 bytes (the number of bits assigned to each label is determined from the size of the corresponding alphabets.)

  When the automaton is an acceptor, only the input label is stored. It is stored in 0 bytes if equal to $\epsilon$; 1 byte, if less than 256; 2 bytes, if between 256 and 65535; 3 bytes, if between 65536 and $2^{24} - 1$.
- $W$ encodes the weight. If zero it is encoded in 0 bytes; otherwise it is stored in 2 bytes, as an integer obtained by linearly scaling between the minimum and maximum weights of the *WFST*.

To access an edge in the chunk, given its offset, all the previous edges in the chunk need to be traversed. In order to alleviate the overhead associated with the expansion and access to the edges, we use a cache that stores the expanded set of edges that leaves the most recently accessed states.

We performed various experiments using various European Portuguese, and North American English, large-vocabulary search networks and language models. We observed that using this representation, we achieved an average of 5.2 bytes per edge when encoding (context-independent) static search networks. When encoding language models, we achieved an average of 7.2 bytes per edge.

## 4. LANGUAGE MODEL OPTIMIZATION

In previous work [5], we observed that most of the gain obtained when globally optimizing the integrated search network ($min\ det(H \circ L \circ G)$) could be achieved by the deterministic composition of a deterministic and minimal language model with a deterministic and minimal lexicon, $H \circ det(min(det(L)) \circ min(det(G)))$. This means that the use of a deterministic and minimal language model is crucial to obtain the best performance with our composition algorithm when embedded in the decoder[3].

[1] A non-coaccessible path is a "dead-end" path that does not reach a final state.

[2] The time spent evaluating the distributions (neural network or gaussian mixtures) is not included in this percentage.

[3] The computational overhead of our implementation of the composition algorithm that also approximates minimization is superior to the gain ob-

```
for all q ∈ Q do
  SetCluster(q, q)
end for
repeat
  nchanges ← 0
  nclusters ← 0
  for all q ∈ Q do
    if cluster(q) = q then
      H[nclusters] ← (q, hash(q))
      nclusters ← nclusters + 1
    end if
  end for
  sort(H)
  for all i = 0..(nclusters − 2) do
    (q_i, h_i) ← H[i]
    j ← i + 1
    (q_j, h_j) ← H[j]
    while (j < nclusters) ∧ (h_i = h_j) do
      (q_j, h_j) ← H[j]
      if (cluster(q_i) ≠ cluster(q_j)) ∧ equiv(q_i, q_j) then
        SetCluster(q_j, q_i)
        nchanges ← nchanges + 1
      end if
      j ← j + 1
    end while
  end for
until (nchanges = 0)
MergeStates()
```

**Fig. 2**. Context-sharing algorithm.

It is thus very important to determinize, push and minimize the language model.

- Determinization can be easily achieved by construction: the language model is approximated by a *WFST* by representing each context by a state and each n-gram by an edge between contexts. The backoffs are represented by edges from specific contexts to more general ones, labeled either with $\epsilon$ or with a special backoff symbol.
- Pushing consists of redistributing the weights as much toward the initial state as possible. Pushing in the language model *WFST* implements a long-range language model look-ahead that goes beyond the next word.
- Minimization of the language model has the very positive effect of sharing contexts. Minimization can be achieved using the standard algorithm for automata, or can be approximated as shown in section 4.1.

### 4.1. Language Model Context Sharing

In this section we present an alternative to the standard automata minimization algorithm for language models represented as *WFSTs*. The main advantage of this *context sharing* algorithm is that it only clusters equivalent states (that represent contexts) without changing the set of edges that leave a particular state. The final topology of the "minimized" language model *WFST* is thus very close to the original one.

tained with the minimization, so, we do not use that version of the algorithm in runtime.

The main idea behind the algorithm is that if two states have the same number of edges and if for every edge leaving one state, there is an equivalent edge leaving the other (that is, with the same input and output labels, the same weight and the same or equivalent destination), then the states are equivalent. The algorithm consists of repeating the comparison of each state with all the others in the *WFST*, until no more equivalent states are found. The complexity of the straightforward implementation of the comparison is quadratic on the number of states of the *WFST*. We circumvent this problem by computing a hash value for each state $q$ (that takes into account the edges that leave the state). That value is stored in a hash array $H$ that contains all pairs $(q, hash(q))$, and is sorted by the hash value. The hash value has the property that it is the same for two equivalent states, but not obligatory different for two different states. To find equivalent states we only need to compare states with the same hash value, which are consecutive on the sorted hash array. The algorithm, described in pseudo-code in figure 2, builds clusters of equivalent states. The clusters are managed with the functions: $SetCluster(q_j, q_i)$ that assigns $q_j$ to cluster $q_i$; $cluster(q)$ that returns the cluster of state $q$; and $MergeStates()$ that replaces each state of the *WFST* with its cluster (merging states as necessary).

One of the design goals of the context-sharing algorithm was to use as little memory as possible. The algorithm, as presented, needs only the space for the *WFST*, the hash array $H$ ($8|Q|$ bytes), and the cluster association vector used by $cluster$ and $SetCluster$ ($8|Q|$ bytes).

## 5. RECOGNITION EXPERIMENTS

### 5.1. Experimental Setup

All the recognition experiments described in this section were based on the broadcast news corpus collected in the scope of the ALERT European project[6]. Their aim is to evaluate the networks produced by our composition algorithm.

The European Portuguese BN corpus collected in this project comprises two subsets: one to be used for speech recognition training and testing and the other to be used for the development of topic detection algorithms.

For the experiments described in this section, we used the training part of the speech recognition corpus (around 60h), and a subset of 256 sentences, randomly selected from the official test corpus. The performance using the 256 utterance subset was actually worse than using the full test set, but its use allowed us to quickly obtain the performance curves needed to compare different optimization schemes.

All the experiments were performed using a standard 1000MHz pentium III PC with 1GB of RAM, running Linux. We used the 1GB of RAM as a resource limit during component development and optimization.

The best acoustic models developed in our research group are based on the combination of the output of various neural networks [7]. In our recognition experiments, we extracted 3 different sets of features from the speech signal: 12 plp coefficients + log energy + deltas; 12 log-rasta coefficients + log energy + deltas; 28 modulation spectrogram features. We used 3 separate multilayer perceptrons (*MLP*), one for each set of features. The input of each *MLP* was a window of 7 vectors centered on the vector being analyzed. The *MLPs* had a 3-layer architecture with 500 units in the hidden layer, and the output consisted of 40 softmax units corresponding to 38 context independent phones plus silence and inspiration noise.

| *WFST* | *edges* | *states* | *compact size* |
|---|---|---|---|
| $det\ G$ | 38,336,415 | 16,878,033 | 282 MB |
| $push\ det\ G$ | 38,336,415 | 16,878,033 | 274 MB |
| $min\ det\ G$ | 26,653,137 | 5,545,173 | 206 MB |
| $min\ push\ det$ | 26,649,611 | 5,543,296 | 198 MB |

**Table 1**. Size of the various LM *WFSTs*.

The output of the 3 *MLPs* was combined using the average of the logarithm of the probability estimated for each phone. The acoustic model topology consisted of a sequence of states with no self-loop to enforce the minimal duration of the model, and one final state with a self-loop. The acoustic models were encoded in a single acoustic model *WFST*.

We used an European Portuguese lexicon with 57k words. The lexicon was converted to a linear lexicon *WFST* with 518,409 states and 584,306 edges. Determinization reduced its size to 157,255 states and 221,321 edges, and minimization allowed an extra reduction to 58,837 states and 121,657 edges.

We used a 4-gram backoff language model, trained from more than 384 million words from newspaper texts and interpolated with models obtained from broadcast news transcriptions. The language model has 5.1M 4-grams, 11M 3-grams, and 5.7M 2-grams. It was converted to a deterministic *WFST* $det\ G$. We also generated a pushed LM *WFST* $push\ det\ G$, by pushing $det\ G$ using the tropical semi-ring. Smaller *WFSTs* were obtained by applying the context-sharing algorithm to $det\ G$ and $push\ det\ G$, resulting in the $min\ det\ G$ and $min\ push\ det\ G$ *WFSTs*. Table 1 shows the size of the various language models.

### 5.2. Results

We investigated the effect of the various possible optimizations of the components on the performance of the system by plotting the word error rate (WER), and the real time speed (xRT), obtained when running it with different beams.

The positive effect of the minimization of the lexicon on the performance of the decoder can be observed in Figure 3. The minimization allows some parcial suffix sharing of the pronunciations of words leading to the same language model state.

We also compared the effect of two major language model optimizations: minimization and pushing. In figure 4, we compare all 4 combinations of pushing and context-sharing minimization. It is patent that context-sharing minimization has very little effect on the performance, while pushing brings some benefits at wider beams, at the cost of much worse performance at narrower beams.

The experiments used the compact representation of *WFSTs* in memory, and where able to run in less than 512MB of RAM.

## 6. CONCLUSIONS AND FUTURE WORK

We have presented techniques that allow the use of the *WFST* approach to large tasks using modest resources. Our dinamic *WFST* composition approach allowed us to improve the speed of our broadcast news system [6], at the same assimptotic error rate, from over 30 xRT to as little as 5 xRT.

The main limitation of this approach is that global *WFST* minimization is not used. In the future, we plan to improve our minimization approximation algorithm [5], and its implementation, in order to further optimize the search network.

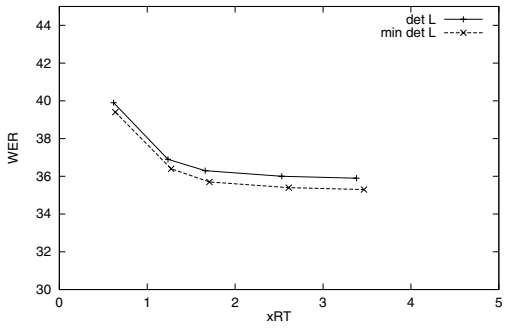


**Fig. 3**. Impact of the minimization of the lexicon *WFST*.

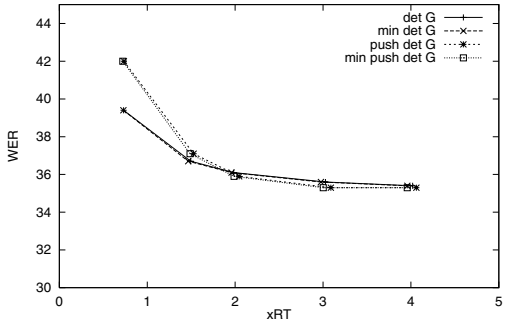


**Fig. 4**. Impact of the optimizations of the language model *WFST*.

## 7. REFERENCES


[1] H. Dolfing and I. Hetherington, "Incremental language models for speech recognition using finite-state transducers," in *Proc. ASRU '2001 Workshop*, Madonna di Campiglio, Trento, Italy, December 2001.

[2] D. Caseiro and I. Trancoso, "On integrating the lexicon with the language model," in *Proc. Eurospeech '2001*, Aalborg, Denmark, September 2001.

[3] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, June 1997.

[4] M. Mohri, F. Pereira, and M. Riley, "Weighted automata in text and speech processing," in *ECAI 96 Workshop*. Budapest, Hungary, August 1996.

[5] D. Caseiro and I. Trancoso, "Transducer composition for "on-the-fly" lexicon and language model integration," in *Proc. ASRU '2001 Workshop*, Madonna di Campiglio, Trento, Italy, December 2001.

[6] H. Meinedo, N. Souto, and J. Neto, "Speech recognition of broadcast news for the european portuguese language," in *Proc. ASRU '2001 Workshop*, Madonna di Campiglio, Trento, Italy, December 2001.

[7] H. Meinedo and J. Neto, "Combination of acoustic models in continuous speech recognition hybrid systems," in *Proc. ICSLP '2000*, Beijing, China, October 2000.